Composer

Composer est un gestionnaire de dépendances externes pour PHP. Il permet de déclarer les bibliothèques et ressources externes nécessaire à un projet, de télécharger et d'installer celles-ci globalement.

Composer fonctionne de manière récursive, ce qui veut dire que si nous déclarons une dépendance qui elle même dépend d'autres ressources, les sous dépendances seront également chargées.

Installation

La manière la plus simple d'installer composer est de télécharger l'archive `composer.phar`.

Pour Windows, il existe également un programme d'installation disponible au téléchargement.

Mise à jour du Path

Il est possible d'utiliser composer en copiant l'archive `composer.phar` à la racine de chaque nouveau projet, mais il est plus judicieux de réaliser une seule installation en copiant ce fichier dans un dossier disponible globalement.

Sous windows

Si nous avons utilisé le programme d'installation Composer-Setup.exe, cette opération n'est pas nécessaire car ce programme s'en est déjà chargé.

Dans le cas d'un téléchargement de `composer.phar`, il faudra enregistrer ce fichier dans un dossier, par exemple `C:\bin` et ajouter ce dossier à la variable d'environnement PATH dans le panneau de configuration système.

Sous Mac OS et Linux

Le plus simple est de compier `composer.phar` dans une arborescence qui est déjà inclue dans les chemins globaux, par exemple `/usr/local/bin/`.
Il est possible d'obtenir la liste des chemins globaux depuis un terminal avec la commande `écho $PATH`.

Pour référencer un nouvel élément dans la variable PATH, il faut modifier le fichier **`/etc/bashrc`** et ajouter la ligne suivante :**`export PATH="$PATH:/nouveau/chemin`**

> ***Note :*** *Chaque chemin est séparé par le caractère* ***:***

Le fichier bashrc définit la configuration des sessions console bash. Il est lu au démarrage d'une nouvelle fenêtre de console. Il faut donc fermer les éventuelles fenêtres console existantes et le rouvrir pour bénéficier de la modification de configuration.

> ***Note :*** *Pour trouver et télécharger certaines dépendances, Composer utilise des commandes Git. Il est donc judicieux que ce logiciel soit également installé sur notre ordinateur.*
> *installer git*

Utilisation

Déclaration des dépendances

Pour utiliser composer, il faut commencer par déclarer les dépendances. Cette déclaration s'effectue dans un fichier composer.json qui contient les paramètres et meta-data de notre projet.
Il est fréquent d'enregistrer ce fichier à la racine du projet PHP.

require

Les dépendances sont déclarées au sein d'un objet `require`qui contient des paires clef valeur où la clef représente le nom de la dépendance et la valeur représente la version attendue.

```
{
    "require": {
        "monolog/monolog": "1.0.*"
    }
}
```

Les versions

Il est possible de spécifier les versions des dépendances de plusieurs façons :

Une valeur exacte

Indique que nous souhaitons télécharger une version spécifique.
exemple : `1.02`

Une plage de valeurs

Indique que nous souhaitons télécharger la version la plus récente dans une plage déterminée.

Les opérateurs de comparaison supportés sont : > < <= >= et `!=`.

Il est possible de définir plusieurs plages en utilisant un séparateur. L'espace et la virgule sont interprétés comme un ET logique. Le double pipe `||` est interprété comme un OU logique.

exemples :
`>=1.0`
`>=1.0 <2.0`

Joker

Le caractère `*` remplace n'importe quelle suite de caractères.

exemple :
`1.*` équivalent à `>=1 <2`

Tilde

Le tilde est un raccourcis qui permet de cibler toutes les versions avant une mise à jour majeure. Ainsi `~1.2` est équivalent à `>=1.2 <2.0`.

Le tilde peut être remplacé par le caractère ^ qui se comporte de façon identique.

Un nom de branche

Il est enfin possible de spécifier un nom de branche plutôt qu'une version numérique. Dans ce cas Composer téléchargera le contenu de cette branche.
Cette dernière option est souvent utilisée lorsque la dépendance ne possède pas de numéro de version. Dans ce cas, la valeur `dev-master` correspond à la branche principal qui est généralement la toute dernière version stable du projet.

Installer les dépendances

Pour installer les dépendances déclarées, il faut lancer une session console depuis le dossier contenant le fichier composer.json et saisir la commande suivante :

```
php composer.phar install
```

Auto chargement

Cette commande installera l'ensemble des dépendances dans un dossier `vendor` à la racine du projet. Composer générera également un fichier `autoload.php` dans le dossier `vendor`. Ce fichier permet le chargement automatique des classes de nos dépendances et nous évite ainsi d'avoir à inclure manuellement toutes les dépendances avec des instructions `include` multiples. Il nous suffira donc l'inclure ce fichier dans notre application pour bénéficier de l'auto-chargement.

Par exemple, imaginons un fichier `index.php` se trouvant dans un dossier `/racine-du-projet/web`

```
include_once __DIR__.'/../vendor/autoload.php;

use Monolog\Logger;
use Monolog\Handler\StreamHandler;

// Instanciation de la class logger
// Inutile d'inclure les dépendances, elles sont auto chargées
$logger = new Logger('mon_logger_perso');
// Définition d'un gestionnaire de log
// ici un fichier .log
$logger->pushHandler(
    new StreamHandler(
        __DIR__.'/app.log',
        Logger::DEBUG
    )
);

// Utilisation du loger
$logger->addInfo('My logger is now ready');
```

Composer.lock

Lors de l'installation des dépendances, Composer génère un fichier `composer.lock` dans lequel il fixe les versions des dépendances. Par la suite, si nous exécutons à nouveau une commande d'installation, ce sont ces versions qui seront utilisées prioritairement par rapport à la déclaration de `composer.json`. L'utilité de ce verrouillage consiste à assurer la compatibilité entre toutes les dépendances même si de nouvelles versions existent qui pourraient mettre en danger cette compatibilité.
Si nous travaillons en équipe avec un outil de contrôle de versions, il faudra soumettre le fichier `composer.lock` de façon à le partager entre tous les développeurs.

Forcer la mise à jour

Pour forcer la mise à jour des dépendances et donc ignorer `composer.lock`, nous utiliserons la commande `update`.

```
php composer.phar update
```

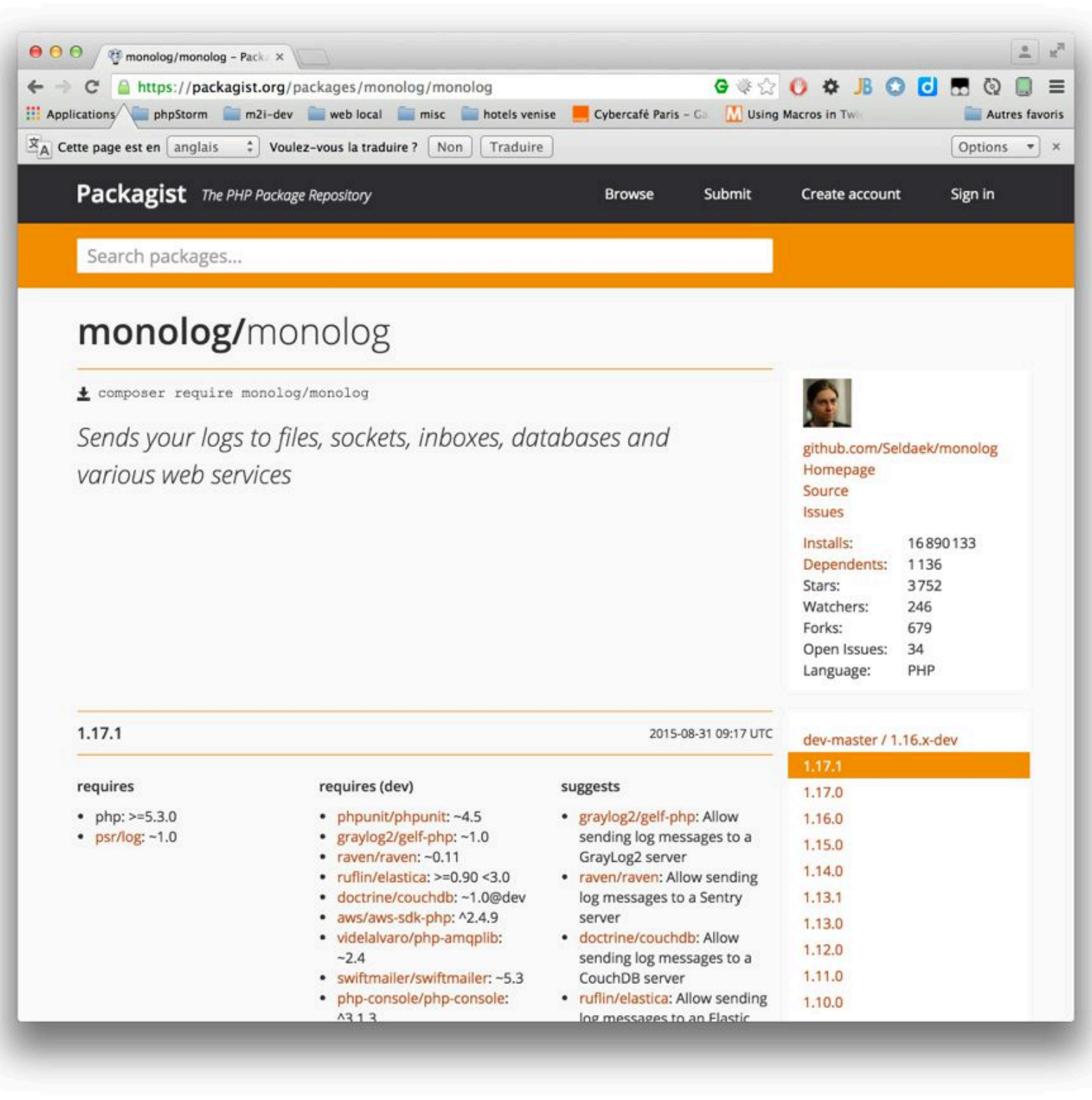
Il est également possible de ne mettre à jour qu'une seule dépendance en précisant son nom après la commande update.

```
php composer.phar update monolog/monolog
```

Rechercher des dépendances

Le site Packagist est la source officielle de dépôt des package PHP utilisable avec Composer. Il est possible d'utiliser d'autres sources, mais il faut alors les déclarer dans `composer.json`.

Le site Packagist permet de recherche un projet et de connaître pour celui-ci les différentes versions disponibles, les dépendances du projet ainsi que d'autres informations telles que les bugs éventuels (issues).



monolog/monolog

composer require monolog/monolog

Sends your logs to files, sockets, inboxes, databases and various web services

github.com/Seldaek/monolog
Homepage
Source
Issues

Installs: 16 890 133
Dependents: 1 136
Stars: 3 752
Watchers: 246
Forks: 679
Open Issues: 34
Language: PHP

1.17.1 2015-08-31 09:17 UTC

requires
- php: >=5.3.0
- psr/log: ~1.0

requires (dev)
- phpunit/phpunit: ~4.5
- graylog2/gelf-php: ~1.0
- raven/raven: ~0.11
- ruflin/elastica: >=0.90 <3.0
- doctrine/couchdb: ~1.0@dev
- aws/aws-sdk-php: ^2.4.9
- videlalvaro/php-amqplib: ~2.4
- swiftmailer/swiftmailer: ~5.3
- php-console/php-console: ^3.1.3

suggests
- graylog2/gelf-php: Allow sending log messages to a GrayLog2 server
- raven/raven: Allow sending log messages to a Sentry server
- doctrine/couchdb: Allow sending log messages to a CouchDB server
- ruflin/elastica: Allow sending log messages to an Elastic

dev-master / 1.16.x-dev
1.17.1
1.17.0
1.16.0
1.15.0
1.14.0
1.13.1
1.13.0
1.12.0
1.11.0
1.10.0

Ajouter une source de package

Imaginons qu'au lieu d'utiliser le package monolog fournit par Packagist, nous souhaitions utiliser un fork de ce projet maintenu par notre ami Igor. Pour ce faire, il nous faut déclarer l'url et le type de dépôt dans une clef `repositories` du fichier composer.json.

```
"repositories": [
        {
            "type": "git",
            "url":  "https://github.com/igorw/monolog"
        }
    ]
```

Documentation complète des repositories

Autoloading

Nous avons vu que Composer générait un fichier `autoload.php` pour assurer l'auto-chargement des classes de nos dépendances. Nous pouvons demander à Composer de faire de même pour nos propres classes et d'intégrer celles-ci à `autoload.php`.

Les options d'auto chargement

psr-0

Cette norme d'autochargement fait correspondre un espace de nom (namespace) avec un chemin racine dans `composer.json`.

```
{
    "autoload": {
        "psr-0": {"Projet\\Modeles\\": "src/"}
    }
}
```

> ***Note 1 :*** *Le double antislash dans les namspace est obligatoire, car ce caractère étant le caractère d'échappement en Json, il doit lui même être échappé.*
>
> ***Note 2 :*** *Il est possible de déclarer de multiples valeurs pour le chemin, dans ce cas on tilisera la notation des tableaux Json [ ].*

Cette configuration déclare l'auto chargement des classes de l'espace de nom `Projet\Modeles` en utilisant un dossier racine `src/`. Cela veut dire que l'autoloader ira chercher les classes dans le chemin suivant : `src/Projet/Modeles/`.

Arborescence du projet

```
- composer.json
[] src
    [] Projet
        [] Modeles
            - Client.php
[]vendor
    - autoload.php
[] web
    - index.php
```

psr-4

Cette norme ressemble beaucoup à psr-0, la différence est que psr-4 n'impose pas la correspondance entre namespace et arborescence. Dans l'exemple ci-dessous, le dossier "Projet" est omis car psr-4 nous permet de définir une correspondance pesronnalisée entre un espace de nom et une arborescence.

```
{
    "autoload": {
        "psr-4": {"Projet\\Modeles\\": "src/Modeles/"}
    }
}
```

Arborescence du projet

```
- composer.json
[]vendor
    - autoload.php
[] src
    [] Modeles
            - Client.php
[] web
    - index.php
```

Classmap

Pour autocharger des classes sans espaces de noms, nous pouvons déclarer un autoloader de type `classmap`, ce module d'autochargement examinera l'ensemble des classes contenues dans les dossiers et fichiers passés en argument et générera un tableau assocatif (le classMap) des classes et de leur chemin.

```
{
    "autoload": {
        "classmap": ["src/", "lib/", "MaClasse.php"]
    }
}
```

Files

Enfin, si nous souhaitons inclure automatiquement des bibliothèques de fonctions (sans classe), nous pouvons utiliser un aoutoloader de type `files`. Attention toutefois, cette opération revient à ajouter un require automatique à tous les fichiers utilisant `autoload.php`. Si la bibliothèque de fonction n'est utilisée qu'occasionnellement dans notra application, il sera plus judicieux de réaliser l'inclusion manuellement.

```
{
    "autoload": {
        "files": ["src/lib/fonctions.php"]
    }
}
```

Mise à jour de l'autoload

Si nous effectuons des modifications au sein de la clef autoload, il faudra mettre à jour `autoload.php` avec la commande suivante :

```
php composer.phar dump-autoload
```